IV International Conference on Computational Methods for Coupled Problems in Science and Engineering
COUPLED PROBLEMS 2011
M. Papadrakakis, E. Oñate and B. Schrefler (Eds)

ALGEBRAIC MULTIGRID PRECONDITIONING FOR MIXED ELLIPTIC-HYPERBOLIC PROBLEMS

Maximilian Emans*

* AVL List GmbH
Hans-List Platz 1, 8020 Graz, Austria
e-mail: maximilian.emans@gmx.at, www.avl.com

Key words: Algebraic Multigrid, Applications, Computing Methods

Abstract. Algebraic multigrid solvers and preconditioners are level of the art solution techniques for many types of linear systems in science and engineering. In this contribution we will compare the computational performance of different algebraic multigrid techniques as preconditioners of Krylov-solvers for coupled systems that reflect the discretisation of problems of mixed elliptic-hyperbolic type. We will report on our experience with different aggregation and cycling strategies as well as on own development and implementation improvements. Our benchmarks are cases of different size from CFD (computational fluid dynamics) applications where the pressure-correction equation is coupled to a transport equation. Very similar systems are those solved in geo-engineering applications, e.g. in oil reservoir simulations. Recently presented k-cycle methods are very efficient and can be readily modified for such linear problems.

1 INTRODUCTION

The efficient solution of linear systems representing the discretisation of mixed elliptic-hyperbolic problems is the kernel of simulations in many areas of science. Some important examples are computational fluid dynamics (CFD), oil reservoir simulations, ground water flow simulations, and semiconductor device simulations. These linear systems reflect the discretisation of coupled differential equations. For subsonic flow problems the pressure usually is ruled by an elliptic differential equation while most of the other variables are essentially determined by equations of hyperbolic type. The common discretisation of problems with both types of equations results in linear systems we refer to as coupled systems.

For large linear problems multigrid techniques are essential parts of effective solvers. In particular for partial differential equations that are discretised on unstructured grids, AMG (algebraic multigrid) methods are attractive since they require for the construction of the grid hierarchy only the information that is stored in the matrix such that the

definition of an interface to the solver part of a program is particularly simple. Since a number of years such solvers are applied with great success to scalar linear problems, i.e. to problems where the components of the unknown vector are associated with a single physical unknown. Sufficiently robust methods are known for a large number of physical problems; appropriate implementations and algorithmic modifications allow to apply algebraic multigrid techniques also on parallel computers with shared or distributed memory without major draw-backs with respect to parallel performance.

For the engineer or scientist who does physical modelling on the level of an own implementation of a simulation program and who wants to solve a particular system, it is a very pleasant fact that a number of very good AMG implementations have been provided to the scientific community through the world wide web. The most prominent examples are hypre, see Falgout and Yang [9], and ML as part of Trilinos, see Gee et al. [10]; rather new is the package AGMG, see Notay [11]. With regard to application to coupled problems and in particular to mixed elliptic-hyperbolic problems, with none of these packages a particularly good performance can be expected. The reason is that for the construction of an appropriate grid the association of the component of the unknown vector to the physical unknowns should be taken into account.

In this contribution we summarise the modifications to existing AMG implementations that are needed for good performance in applications where mixed elliptic-hyperbolic problems are to be solved. We then focus on aggregation AMG and compare the performance of solvers that are obtained by adapting the most important aggregation algorithms to coupled problems.

2 ALGEBRAIC MULTIGRID FOR COUPLED SYSTEMS

We consider a continuous problem with more than one unknown function, i.e. with more than one physical unknown or independent variable. The discretisation leads us to a system of linear equations. Let us denote the linear problem for the moment as

$$A\vec{x} = \vec{b} \tag{1}$$

where $A \in \mathbb{R}^{n\times n}$ is regular and sparse, $\vec{b} \in \mathbb{R}^n$ is some right-hand side vector and $\vec{x} \in \mathbb{R}^n$ the solution where n is the rank of A and thus the number of components of the unknown solution vector $\vec{x}$. These components are referred to as "nodes" in a large part of the literature on AMG. One node corresponds to one unknown value of one of the function that are approximated by the solution of the linear system. It refers to the vertices of the connectivity graph of the matrix and not to a geometric entity (like a point of the discretisation scheme). Note that for coupled systems these difference is substantial while for scalar systems it is only a question of the word choice.

An AMG algorithm comprises two phases: A setup phase and a solution phase. In the setup phase the operators $A_k \in \mathbb{R}^{n_k\times n_k}$ $(k = 2, ..., k_{max},\ A_k = A)$ with system size n_k are defined where $n_{k+1} < n_l$ holds for $k = 1, ..., k_{max} - 1$. Moreover, a smoother S_k as well

as a prolongation operator P_k and a restriction operator R_k have to be determined for each level k. The effects of prolongation and restriction are illustrated in figure 1. It is common practice in algebraic multigrid to choose $R_k = P_k^T$ and to follow the Galerkin approach to generate the coarse-grid hierarchy recursively by

$$A_{k+1} = P_k^T A_k P_k \qquad (k = 1, ..., k_{max} - 1). \tag{2}$$

Various ways have been described in the literature to define P_k; in fact this is the major difference between the different AMG methods.

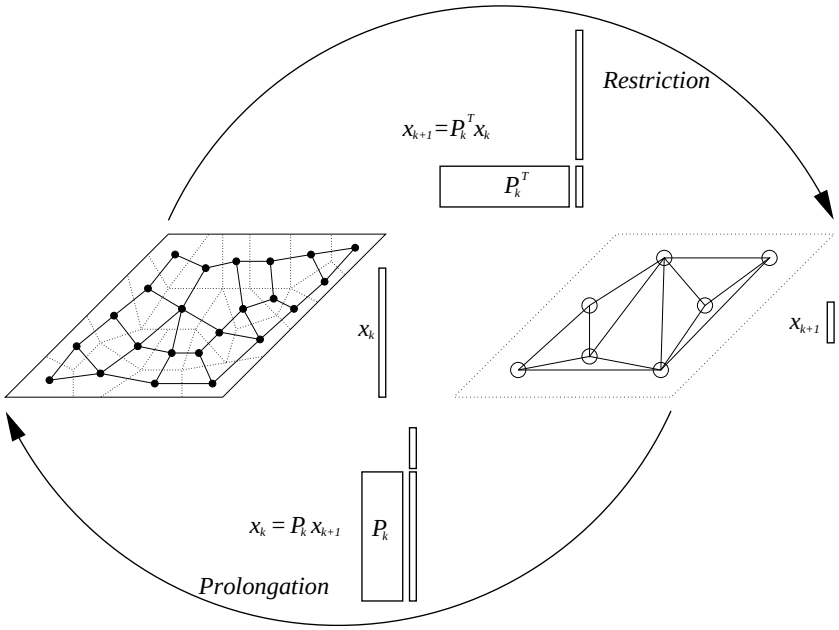

Figure 1: Illustration of prolongation and restriction

The prolongation operator P_l projects a vector of size n_{l+1} which represents a solution on level $l+1$ to a vector of size n_l representing a solution on level l. Generally, the values on level l are interpolated values where the supporting points are the nodes on level $l+1$. In a geometric scheme the interpolation relies on the spatial relation of the nodes; in an algebraic scheme, however, the geometric relation of the nodes is not explicitly known and is therefore replaced by the matrix connectivity. Note that in many cases, on the finest level, the geometric relation is reflected by the matrix connectivity. It has to be avoided that the interpolation takes place between two nodes associated with different physical values. For AMG schemes it is therefore not enough to consider only the information contained in the matrix: those connections corresponding to edges in the connectivity graph that link nodes associated with different physical unknowns need to be skipped in the construction of the interpolation scheme. Assuming (for simplicity) an ordering of the

Figure 2: Prolongation operators assuming ordering of nodes by physical unknown (left: scalar systems, right: coupled systems, grey area: first physical unknown out of two)

nodes by physical unknown, this leads to a block structure of the prolongation operators as shown in figure 2.

The blocks in the prolongation operator can now be constructed in exactly the same way as the corresponding operators for scalar systems. The association of the nodes to the physical unknown can either be tracked by a marker field or, as we did in the implementations used for this work, the variables are ordered by physical unknown on each grid; in our opinion the latter way is advantageous since practically no reordering on the coarse-grid is necessary because it is natural to define the coarse-grids for the individual physical unknowns one after the other.

If indeed the coarse-grid of each physical unknown is constructed as if the matrix would describe a scalar problem, then the method is referred to as unknown-based approach, see Clees and Ganzer [3]. The same authors report that the employment of an identical aggregation scheme to all physical unknowns can be beneficial for the convergence. In terms of a geometrical method this approach means that for each physical unknown on any point the same coarse-grid points are involved in the interpolation. This scheme is referred to as point-based approach. Note that this cannot be exploited to simplify the computation of the coarse-grid operators by Galerkin approach (2).

In the solution phase these structures are employed to approach iteratively, starting from an initial guess, the solution. The pseudocode of a standard v-cycle algorithm as one of the easiest cycling strategies is shown in algorithm 1. In practice, the cycling strategy, the smoother and its parameters, and the choice of the prolongation and restriction operator should be adjusted carefully to each other in order to obtain an efficient method.

3 BENCHMARKS

In this section we first describe shortly five algorithms; the selection reflects technically feasible possibilities. For details we will refer the interested reader to the literature. For

Algorithm 1 v-cycle AMG

$\vec{x}_k^{(3)} = \text{AMG\_par}(k, \vec{r}_k, \vec{x}_k^{(0)})$
Input: level l, right-hand side $\vec{r}_k$, initial guess $\vec{x}_k^{(0)}$
Output: approximate solution $\vec{x}_k^{(3)}$

1: **if** $k = k_{max}$ **then**
2: solution of coarse-grid system: $\vec{x}_k^{(3)} = A_k^{-1}\vec{r}_k$
3: **else**
4: pre-smoothing: $\vec{x}_k^{(1)} = S_k(\vec{r}_k, A_k, \vec{x}_k^{(0)})$
5: restriction: $\vec{r}_{k+1} = P_k^T(\vec{r}_k - A_k\vec{x}_k^{(1)})$
6: recursive solution of coarse-grid system: $\vec{x}_{k+1} = \text{AMG\_par}(k+1, \vec{r}_{k+1}, \vec{0})$
7: prolongation of coarse-grid solution and update: $\vec{x}_k^{(2)} = \vec{x}_k^{(1)} + P_k\vec{x}_{k+1}$
8: post-smoothing: $\vec{x}_k^{(3)} = S_k(\vec{r}_k, A_k, \vec{x}_k^{(2)})$
9: **end if**

the AMG variants we name sources of efficient implementations of the algorithms. Later on we present two benchmarks: the sample systems are two linear systems that reflect the discretisation of a problem of elliptic-hyperbolic type from a CFD application.

3.1 AMG variants

The smoother in all algorithms is a ILU(0) smoother with one sweep before and after the coarse-grid correction. The coarse-grid treatment is done by an agglomeration scheme, i.e. as soon as the grid on one of the processes becomes too small in the course of the coarsening, it is merged to the grid of a neighbour; whenever in the solution phase action on this or a coarser level is taken, this process is idle. The threshold grid size is 200 nodes. An alternative to this procedure is the employment of a sparse direct solver such as MUMPS, see Amestoy et al. [1]. It is easier to implement, but in particular for smaller problems or for runs on distributed systems with slow interconnect the computation might take up to 20 % more time, see Emans [6]. If one starts with one of the mentioned packages, the question which coarse-grid treatment technique to choose is obsolete since these packages cover the parallelisation of the whole algorithm, e.g. AGMG uses MUMPS. The differences between the algorithms are the coarse-grid selection schemes and, related to this, the cycling strategies. This will be outlined in the following.

Smoothed Aggregation AMG This scheme forms aggregates containing typically between 20 and 50 nodes. Formally, a tentative prolongation operator with constant interpolation is constructed. Since it would result in rather poor representation of the fine-grid problem on the coarse-grid, it is smoothed by applying one Jacobi-smoothing step to it. The implemented serial algorithm has been described by Vaněk et al. [15]. In our parallel implementation the aggregation and the smoothing are strictly local processes, i.e. aggregates are not intersected by inter-domain boundaries. More details of the parallel

implementation used here have been discussed in the previous publication [5]. Another similar implementation of this algorithm can be found under the short ML in the solver package Trilinos [10]. A measure for the memory requirement is the operator complexity, i.e. the ratio of the sum of the number of nodes on all grids and the number of nodes on the finest grid. For this method it is typically around 1.5. Here, this algorithm is used as a preconditioner of a GMRES method. The multigrid cycle is a v-cycle with one pre-smoothing and one post-smoothing sweep of the ILU(0) smoother. The algorithm is referred to as **ams1gm**.

Pairwise Aggregation AMG Notay [11] suggested recently an efficient algorithm that produces aggregates of only two nodes. Due to the small aggregate size, a solution on grid $k+1$ is represented with reasonable quality on grid k if constant interpolation is employed. With this interpolation scheme, the prolongation operator P_k is also particularly simple: One row (that corresponds to a node on level k) contains exactly one entry with value one, namely in the column that corresponds to the aggregate on level $k+1$ this node is assigned to; all other elements are zero. The computation of the coarse-grid hierarchy is particularly cheap: Equation (2) reduces essentially to an addition of rows of A_k. Consequently, the computation of the coarse-grid hierarchy in total is inexpensive. It is local to each process since only nodes assigned to the same process are grouped together in an aggregate. However, the number of grids will be rather large since from level l to level $l+1$ the number of nodes is reduced only by a factor of approximately two. It can be seen as a disadvantage that the operator complexity of this method is about 2.5. The cycling strategy is an f-cycle, see Trottenberg et al. [14] for a definition, with one pre-smoothing and one post-smoothing sweep of the ILU(0) smoother. The algorithm is referred to as **amf1gm**.

GCR-accelerated AMG This kind of preconditioner has been suggested by Notay [11]. An implementation (which is quite similar to ours) is available as AGMG from the homepage of the author. The main difference to conventional (fixed cycle) AMG preconditioners is that the coarse-grid system is approximated by one or two iterations (depending on a termination criterion) of an inner GCR-solver [4] (in a modified economic implementation) that is preconditioned by AMG instead of being approximated by the multigrid scheme alone. Since this preconditioning operation is not the same in each iteration, standard Krylov-methods cannot be used as outer iteration. For nonsymmetric problems Notay [11] uses GCR also as outer Krylov-solver. In this algorithm the double-pairwise aggregation suggested by Notay [11] is employed: It is obtained by applying the pairwise aggregation of algorithm amf1gm twice. The aggregates usually comprise four nodes, the computation of the coarse-grid system is similarly cheap as that of amf1gm. Since only every second grid and its associated operator is stored, the operator complexity is only about 1.6 for this method. We refer to this algorithm as **amk1gm**.

GMRES-accelerated AMG In this algorithm the GCR in amk1gc as inner and outer solver is replaced by GMRES. Since the preconditioning operation is not the same in each iteration, GMRES is implemented as FGMRES, see Saad [12]. In our experience this GMRES-based method is significantly more robust than the GCR-based one. A discussion of this issue along with relevant examples can be found in Emans [7]. All other parts of the algorithm, in particular the cycling strategy and the coarsening algorithm, are the same as that of amk1gc. A potential disadvantage of this method is that (up to the knowledge of the author) an implementation cannot be directly obtained and consequently some implementation effort will be faced. We refer to this algorithm as **amk1gm**.

ILU-preconditioned BiCGstab The algorithm that serves as a reference is a ILU(0)-preconditioned BiCGstab method. It has been deduced and discussed in detail by Saad [13]. Our implementation requires only to store one additional vector of the size of the unknown vector. We apply two sweeps in each iteration of the Krylov method. Here, we refer to this method as **ilu0bc**.

3.2 Performance

In order to compare the efficiency of the algorithms we have chosen two linear problems from different CFD applications. The SIMPLE algorithm extended by a pressure-enthalpy coupling, see Emans et al.[8], leads to a system of the type

$$\begin{pmatrix} C & S_h \\ S_p & G \end{pmatrix} \begin{pmatrix} \vec{p}\,' \\ \vec{h}' \end{pmatrix} = \begin{pmatrix} \vec{c} \\ \vec{g} \end{pmatrix} \tag{3}$$

where C is the symmetric positive definite operator of the pressure-correction equation, G the operator of the transport equation for the enthalpy, S_h and S_p are the coupling operators, $\vec{c}$ and $\vec{g}$ are the right-hand sides; the components of the unknown vector are either associated with the pressure correction $\vec{p}\,'$ or with the enthalpy update $\vec{h}'$. Since the underlying partial problem of the pressure-correction equation is elliptic and the transport of the enthalpy is essentially a hyperbolic problem, the system is referred to be of mixed elliptic-hyperbolic type. The iterative solution procedure is terminated if the 1-norms of the residuals of both parts of the system have been reduced by a factor of $1.0 \cdot 10^{-6}$.

In problem 012 the flow of cold air (293 K) into a complex engine cylinder geometry with hot walls (650 K) is simulated; the mesh consists of 1.4 mio cells of which approximately 80% are hexagonal while the rest is tetrahedral. In problem 045 a backward-facing step problem, see Armaly et al. [2], is solved where hot gas (1100 K) is entering the domain that is initially filled by cold gas (293 K). The orthogonal mesh of this case has 2.3 mio cells.

For the benchmarks we used up to eight nodes à 2 quad-cores of a Linux cluster; each node of this cluster is equipped with two Intel Xeon CPU X5365 (3.00GHz, main memory 16GB, L2-cache 4MB shared between two cores). The nodes are connected

by an Infiniband interconnect with an effective bandwidth of around 750 Mbit/s and a latency of around 3.3 μs. The computational part of the program is compiled by the Intel-FORTRAN compiler 10.1, the communication is performed through calls to hp-MPI subroutines (C-binding). The benchmarks were run within the environment of the software AVL FIRE$^{(R)}$ 2009 with 1, 2, 4, 8, 16, and eventually 32 processes. Computations with 1, 2, and 4 processes were done on a single node, for more processes we used 2, 4, and 8 nodes respectively. This is done to ensure that each process has sole access to one of the L2-caches on the node. It is important to note that the MPI implementation uses the shared memory space of one node for the communication between two processes wherever possible. This means that all intra-node communication (calculations with up to 4 processes) is done without utilisation of the network interconnect.

In figures 3 and 4 the number of iterations, the computing times, and the parallel efficiency $E_p = t_1/(t_p \cdot p)$ (t_p: computing time with p parallel processes) are shown. First it is interesting to see if and at what rate the numerical method converges. Convergence within 250 iterations is observed in all AMG calculations apart from the ams1gm runs for problem 012. In this case the algorithm does neither stall nor diverge, but the convergences is so slow that the criterion is not reached within 250 iterations. The diagrams with the iteration count shows further that the AMG methods converge much more rapidly than the ILU(0) method. Apart from ams1gm for problem 045, the AMG solvers the parallelisation does not mitigate the convergence behaviour.

The computing times show that the k-cycle methods (amk1gc and amk1gm) are the fastest methods. They are faster by 50% and by 90% compared to ilu0bc for 012 and 045 respectively. The larger advantage in the latter case is due to the problem size. Compared to the pairwisely aggregating AMG amf1gm the advantage is still considerable. Comparing now amk1gc and amk1gm the results show that the differences are rather small: for problem 012 amk1gm is slightly faster, for problem 045 amk1gc is slightly faster. Since both systems here are solved by both algorithms, the main advantage of amk1gm stays hidden: It is its larger robustness, see Emans [7]. From the view point of practical usage, amk1gc is advantageous since, coming from AGMG, only the modification of the coarse-grid generation for coupled system needs to be implemented.

The parallel efficiency of both algorithms amk1gc, amk1gm, and amf1gm is satisfying and competitive to that of ilu0bc. The curves are typical for computations on the used hardware: For up to four processes a decrease is observed. This is due to the fact the the data transfer mechanism on the chip which is a Front-Side Bus (FSB) is the bottleneck; the data transfer between the processes is fast since it relies on the shared memory mechanism. If more processes are involved, inter-node data transfer through the interconnect is required; more important is that the probability of cache misses is reduced since the decomposed problems have become smaller and a larger portion of them fits into the cache. Consequently the curves rise. The rise is more pronounced for smaller problem 012. The observed parallel efficiencies of these coupled solvers correspond well to the parallel efficiencies which can be reached with similar methods for scalar problems.

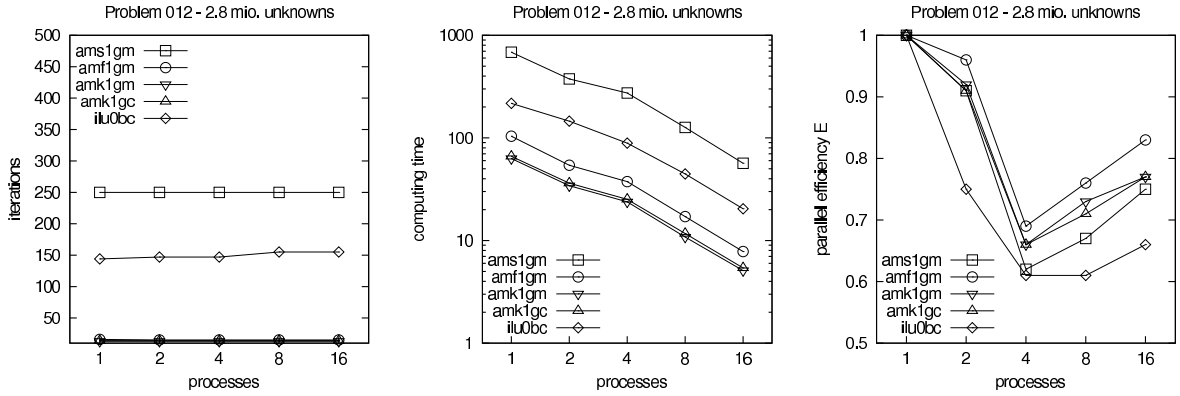

Figure 3: Iteration count (left), computing times (middle), and parallel efficiency (right) for benchmark 012

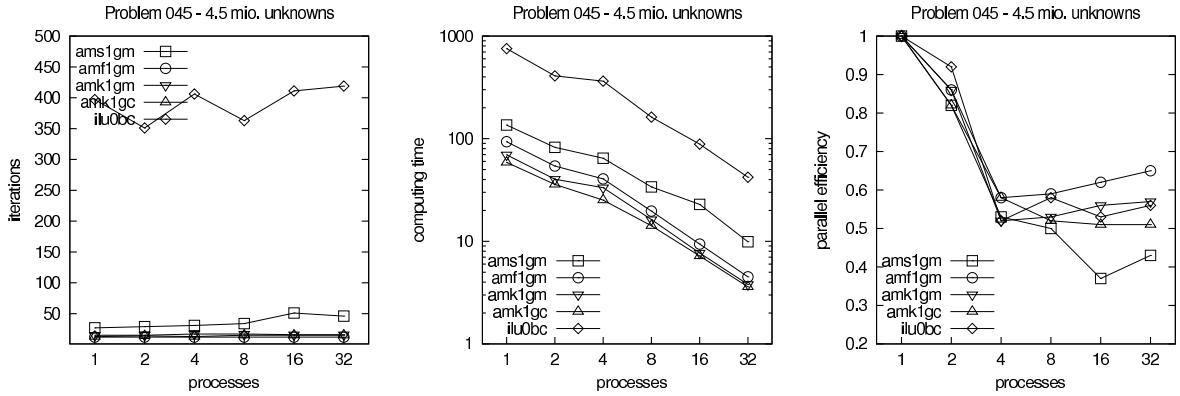

Figure 4: Cumulative iteration count (left), computing times (middle), and parallel efficiency (right) for benchmark 045

4 CONCLUSIONS

The k-cycle methods amk1gc and amk1gm are the fastest AMG solvers for our benchmarks. Practically, amk1gc can be deduced from existing methods which are available to the scientific community where the effort to implement the modifications is small. These relatively new algorithms outperform both, Smoothed Aggregation AMG and ILU(0)-preconditioned BiCGstab.

REFERENCES

[1] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23:15–41, 2001.

[2] B.F. Armaly, F. Durst, J.C.F. Pereira, and B. Schonung. Experimental and theoretical investigation of backward-facing step flow. *Journal of Fluid Mechanics*, 48:473–

496, 1983.

[3] T. Clees and K. Stüben. Algebraic multigrid for industrial semiconductor device simulation. In E. Bänsch, editor, *Challenges in Scientific Computing – CISC2003*, volume 35 of *Lecture Notes in Computational Science and Engineering*, pages 110–130. Springer-Verlag, 2002.

[4] C.S. Eisenstat, H.C. Elman, and M.H. Schultz. Variational iterative methods for nonsymmetric systems of linear equation. *SIAM Journal on Numerical Analysis*, 20:345–357, 1983.

[5] M. Emans. Approximate solutions of linear systems in CFD applications. *SIAM Journal on Scientific Computing*, 32:2235–2254, 2010.

[6] M. Emans. Coarse-grid treatment in parallel AMG for coupled systems in CFD applications. 2010. submitted.

[7] M. Emans. K-cycle AMG for semi-definite and nonsymmetric systems in CFD. 2010. submitted.

[8] M. Emans, S. Frolov, B. Lidskii, V. Posvyanskii, Z. Žunič, and B. Basara. Pressure-enthalpy coupling for subsonic flows with density variation. In M. Rahman and C.A. Brebbia, editors, *AFM VIII*, pages 127–136. WIT Press Southampton, 2010.

[9] R.D. Falgout and U.M. Yang. hypre: a library of high performance preconditioners. In P.M.A. Sloot, C.J.K. Tan, J.J. Dongarra, and A.G. Hoekstra, editors, *ICCS2002, Part III*, volume 2331 of *Lecture Notes on Computer Science*, pages 632–641. Springer, 2002.

[10] M.W. Gee, C.M. Siefert, J.J. Hu, R.S. Tuminaro, and M.G. Sala. *ML 5.0 Smoothed Aggregation User's Guide*. SAND2006-2009 Unlimited Release, 2006.

[11] Y. Notay. An aggregation-based algebraic multigrid method. *Electronic Transactions on Numerical Analysis*, 37:123–146, 2010.

[12] Y. Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM Journal on Scientific Computing*, 14:461–469, 1993.

[13] Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, 3 edition, 2003.

[14] U. Trottenberg, C. Oosterlee, and Anton Schüller. *Multigrid*. Elsevier Academic Press Amsterdam, 2001.

[15] P. Vaněk, J. Mandel, and M. Brezina. Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems. *Computing 56*, pages 179–196, 1996.